VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Modulární systém pro předzpracování dat

Modular system for data preprocessing

2013 **Ondrej Papaj**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Ondrej Papaj**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Modulární systém pro předzpracování dat
Modular System for Data Preprocessing

Zásady pro vypracování:

Cílem práce je navrhnout modulární systém pro inteligentní předzpracování dat a to tak, aby jednotlivé použitelné moduly byly realizovány jako pluginy. Systém musí být schopen pracovat i s velmi velkými daty, musí být schopen ukládání scénářů zpracování, které bude možné aplikovat na různé vstupní soubory, musí uchovávat historii provedených změn pro zpětnou rekonstruovatelnost, a dále také musí uchovávat metadata o již analyzovaných datech. V neposlední řadě musí být systém uživatelsky příjemný a snadno použitelný.

Práce bude obsahovat:
1. Analýzu řešeného problému a popis všech aspektů řešení.
2. Návrh rozhraní pro pluginy a rozbor požadavků.
3. Návrh implementace jednotlivých částí aplikace.
4. Implementaci aplikace a pluginů.
5. Testování aplikace.

Aplikace bude implementována pomocí platformy .Net a jazyka C#.

Seznam doporučené odborné literatury:

[1] Ján Hanák: Praktické objektové programování v jazyce C# 4.0, 978-80-87017-07-4, Artax 2009
[2] Ján Hanák: Praktické paralelní programovaní v C# 4.0 a C++, 978-80-87017-06-7, Artax, 2009
Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Platoš, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlásenie študenta

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

Dňa: 06.05.2013 .....

Poďakovanie

Týmto by som sa rád poďakoval hlavne pánovi Ing. Janu Platošovi PhD. za odbornú pomoc a konzultácie pri vytváraní tejto bakalárskej práce. Poďakovanie patrí aj mojej priateľke za psychickú podporu.

Abstrakt

Cieľom tejto bakalárskej práce je oboznámenie s procesom predspracovania dát. Prvá časť popisuje jednotlivé metódy predspracovania. V druhej časti je implementácia týchto metód a vytvorenie nástroja, ktorý je schopný pripraviť dáta na následné spracovanie.

Kľúčové slová

Predspracovanie dát, plugin, atribút, metadáta

Abstract

The goal of this thesis is to inform about the process of data preprocessing. The first part describes methods of preprocessing. The second part contains the implementation of these methods and also the implementation of the tool which prepares input data for further processing.

Keywords

data preprocessing, plugin, attribute, metadatas

Zoznam použitých skratiek

CSV – comma-separated values - hodnoty oddelené čiarkami) je jednoduchý súborový formát vo forme čistého textu (angl. plain text) určený na ukladanie tabuľkových dát.

ARFF – attribute relation file format – súbor, ktorý vyjadruje vzťahy medzi atribútmi, vyvnutý na univerzite Waikato.

ASCII - je anglická skratka pre American Standard Code for Information Interchange (americký štandardný kód pre výmenu informácií). Ide o kódovací systém znakov anglickej abecedy, číslic, iných znakov a riadiacich kódov

Unicode - Unicode je medzinárodný štandard, ktorého cieľom je definovať kódovaciu schému schopnú reprezentovať väčšinu znakov používaných v písaných jazykoch spolu s inými symbolmi.

Zoznam použitých termínov

.NET - .NET Framework – je softvérový rámec vyvinutí spoločnosťou Microsoft.

Microsoft Visual Studio 2012 – vývojové integrované prostredie od spoločnosti Microsoft.

C# - vysokoúrovňový objektovo-orientovaný programovací jazyk vyvinutý firmou Microsoft s platformou .NET Framework.

Zoznam obrázkov

Obsah

1 Úvod

V dnešnej dobe sme priam zahltení dátami. Podľa prieskumu ich priemerný človek za rok vyprodukuje až 333MB [1]. Ich problém je, že pochádzajú z rôznych zdrojov a ani jeden z týchto zdrojov nie je na 100% spoľahlivý. To znamená, že v týchto dátach sa nachádzajú chyby, ktoré treba eliminovať. Ak by sa tak nestalo, konečné výsledky pri modelovaní by mohli byť úplne odlišné ako pri predspracovaných dátach. Rastúce množstvo dát, ktoré je hlavným dôsledkom moderného „process monitoring“-u a systému zbierania dát, vyžaduje používanie efektívnych techník predspracovania [2]. Ďalším dôvodom na predspracovanie je, že systém, ktorý bude s dátami pracovať ich vyžaduje v inej podobe, a preto ich treba nejakým spôsobom upraviť.

Najlepšou cestou ako sa s procesom predspracovania a s princípom jeho metód zoznámiť je ich praktická implementácia. Na začiatku práce je popísaná činnosť a vlastnosti funkcií na predspracovanie a v ďalších častiach ich samotná implementácia v systéme.

2 Metódy na predspracovanie dát

S dátami reálneho sveta je vždy nejaký problém. Charakter problémov je taký, že sa vyskytujú aj napriek ľudskej snahe ich eliminovať. Problémy s dátami môžeme členiť do troch hlavných skupín :

1. **príliš veľa dát** – tento problém nám sťažuje modelovanie výsledku

2. **príliš málo dát** – opačný problém ako v prvom prípade, z malého množstva dát nemusí byť výsledok najpresnejší

3. **poškodené dáta** – dáta sú získané z viacerých zdrojov alebo nemajú korektný formát

2.1 Metóda na odstránenie chýbajúcich hodnôt

Dôvod chýbajúcich hodnôt máva rôzne príčiny. Môže to byť napríklad rozbitý senzor, alebo ľudia, ktorí odmietli alebo zabudli vyplniť dotazník alebo nejaký atribút nemá význam pre isté druhy objektov (napríklad gravidita nemá u samcov význam). Je veľmi dôležité identifikovať typ *„skrytej“* chýbajúcej hodnoty, lebo následná analýza inak môže viesť k nezmyselným záverom. Najčastejšie sa u chýbajúcich hodnôt predpokladá, že dáta chýbajú úplne náhodne (*Missing Completely at Random)* [3]. Naopak chybový model *Missing at Random* [3] pracuje s predpokladom, že pravdepodobnosť chýbajúcej hodnoty závisí na nejakej inej premennej. Najhorším prípadom sú neignorovateľné chýbajúce hodnoty (*Nonignorable Missing Values*) [3].

Chýbajúce hodnoty nie je možné vždy vylúčiť. Najjednoduchší spôsob je nahradiť hodnotu u numerického typu priemerom alebo hodnotou, ktorá má najčastejší výskyt u atribútu kategoriálneho tak, ako to robí systém tejto práce.

Ďalší spôsob je, že nahradíme chýbajúcu hodnotu najbližším susedom, teda hodnotou, ktorá je najpodobnejšia tej chýbajúcej. Tu ale hrozí, že nechtiac vytvoríme nejakú závislosť medzi prvkami.

2.2 Normalizačná metóda

Touto metódou sa snažíme eliminovať nerovnaký vplyv alebo dôležitosť premenných. Nemusí vždy platiť, že vysoká hodnota ma väčšiu funkčnú hodnotu. Toto vyriešime pomocou normalizácie.

Existuje niekoľko druhov normalizácie, ktoré si v ďalšej časti predstavíme.

2.2.1 **Lineárna transformácia**

Lineárna transformácia, taktiež nazývaná min-max normalizácia, je použitá aj v systéme tejto bakalárskej práce. Najčastejšou úpravou dát je ich transformovanie do intervalu <0,1> a to podľa vzorca [4]:

$$x \rightarrow \frac{x - min_x}{max_x - min_x}$$

kde min_x a max_x sú minimum a maximum daného atribútu. Následný prevod z tohto intervalu na iný už je jednoduchý pomocou rovnice :

$$x \rightarrow \frac{x - min_x}{max_x - min_x} \ .(O_{max} - O_{min}) + \ O_{min}$$

kde O_{max} , O_{min} sú hranice nového intervalu.

Nevýhodou tejto normalizácie je, že väčšina rozsahu môže ostať nevyužitá, a to v prípade, že atribút obsahuje výrazne väčšie alebo výrazne menšie hodnoty ako je stredná hodnota. V prípade ak sú tieto hodnoty výsledkom nejakej anomálie, spôsobí to skreslenie informácií.

2.2.2 **Z-score normalizácia**

Táto metóda sa používa často v štatistike na štandardizáciu dát. Z-score vyjadruje vzdialenosť hodnoty od strednej hodnoty danej množiny prvkov atribútu. Táto vzdialenosť je v tomto prípade vyjadrená násobkom veľkosti výberovej smerodajnej odchýlky.

Strednú hodnotu pre daný atribút získame tak, že vydelíme súčty výsledkov pozorovaní počtom týchto pozorovaní. Stredná hodnota udáva centrálne umiestnenie a odchýlka udáva rozptyl daných dát [5]:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Ako príklad možno uviesť príklad o priemernej teplote na dvoch miestach s rôznymi klimatickými podmienkami. Priemerná teplota môže byť rovnaká ale odchýlka vo vnútrozemí bude väčšia ako na pobreží. Pre Z-score normalizáciu ešte potrebujeme vypočítať smerodajnú odchýlku

$$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

kde $\bar{x}$ je aritmetický priemer hodnôt atribútu. Tento vzorec však vyžaduje predbežný výpočet $\bar{x}$, čo znamená jednu iteráciu cez všetky hodnoty.

Takže vstupné dáta normalizujeme tak, že od každej hodnoty odpočítame strednú hodnotu vstupných dát a potom výsledok vydelíme veľkosťou smerodajnej odchýlky:

$$z = \frac{x - \bar{x}}{\sigma}$$

2.2.3 Transformácia strednej hodnoty

Cieľom tejto metódy je určiť nulovú strednú hodnotu vstupných dát. Toto môže byť užitočné v prípade, že máme viac zdrojov porovnateľných dát so vzájomným offsetom. Ako príklad môžeme uviesť dáta z niekoľkých teplomerov, ktoré nie sú dobre skalibrované, ale správne vyjadrujú hodnotu rozdielu meraných teplôt. Normalizácia potom prebieha takýmto spôsobom:

$$x' = x - \bar{x}$$

2.3 Odstránenie odľahlých hodnôt

Odľahlé hodnoty bývajú v zásade spôsobené chybným meraním alebo preklepom. Takéto chybné dáta , v prípade, že nejde urobiť nápravu, je potrebné vylúčiť. Samozrejme, že nejaká odľahlá hodnota môže byť správna a jej odlišnosť môže byť daná nejakou výnimočnou situáciou. Aj napriek tomu, že hodnoty sú správne, môže (ale aj nemusí) byť vhodné ich vylúčiť. Existujú analytické metódy, ktoré sú proti takýmto odľahlým hodnotám robustné. Ale niektoré metódy sú tak citlivé, že aj jedna takáto hodnota môže viesť k nezmyselným výsledkom.

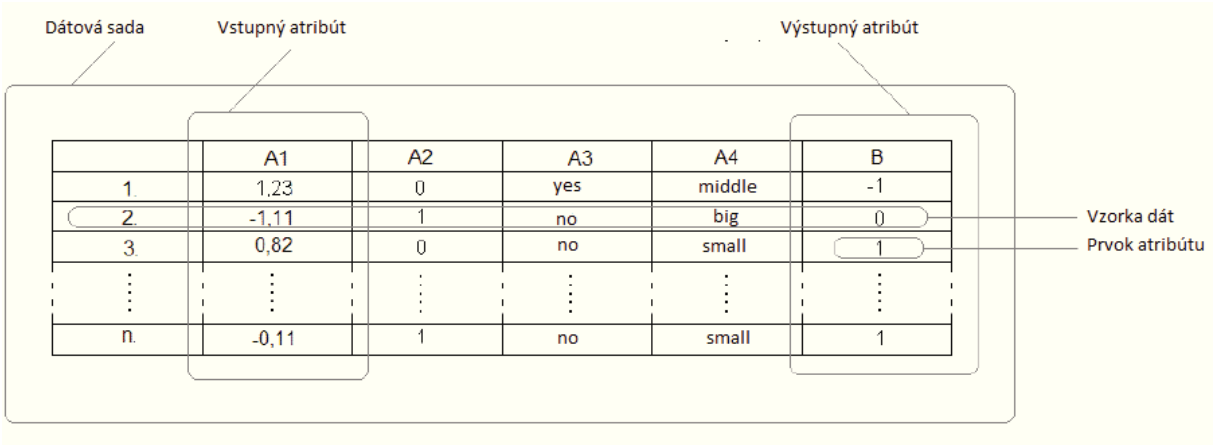
Odľahlé hodnoty závisia na distribučnej funkcii, z ktorej boli dáta navzorkované. Pri detekcii takýchto hodnôt pomocou jedného atribútu sa u kategorických dát vychádza z toho, že sa odľahlé

hodnoty vyskytujú veľmi zriedkavo. U numerických typov je situácia komplikovanejšia. U väčších dátových súborov existujú vždy nejaké hodnoty, ktoré budú odľahlé.

Pri normálnom rozložení hodnôt leží 95 % hodnôt vo vzdialenosti dvakrát veľkosť smerodajnej odchýlky. Hodnoty veľmi vzdialené od priemernej hodnoty spôsobujú veľké chyby pri spracovaní a môžu mať katastrofálne následky. Ak je ich počet malý, hodnoty sa vo väčšine prípadov vypúšťajú. Pokiaľ je ich počet veľký, nie je možné tieto hodnoty ignorovať, treba použiť metódy, ktoré nie sú citlivé na odľahlé hodnoty. Veľmi citlivou je napríklad metóda najmenších štvorcov [4].

2.4 Funkcie pracujúce s prvkami atribútov

Aby sme lepšie porozumeli ich významu, zjednotíme si použité pojmy a vytvoríme si prehľad. Na tomto obrázku sú vysvetlené všetky pojmy, ktoré sú pre naše pochopenie potrebné:



Obrázok 1: Súčasti dátovej sady

Dátová sada je súbor všetkých dát. Vstupný atribút je nezávislý a je vyjadrený buď číslom (numerický) alebo popisom (nominálny). Výstupný atribút závisí na vstupných atribútoch. Jeden riadok sa nazýva vzorka dát a vyjadruje, za akých podmienok má výstupný atribút danú hodnotu. Jeden prvok je jedna hodnota z atribútu.

2.4.1 Diskretizácia dát

BinDiscretization - táto funkcia rozdelí dáta na triedy. V tomto prípade je počet tried vstupný atribút funkcie. Nájdeme maximum a minimum, ktoré od seba odčítame a vydelíme počtom tried, dostaneme tak *i* . Rozdelíme dáta na dve množiny. V prvej budú dáta <minimum-i> a v druhej (i-2i> …(xi-max> .[6]

Príklad:

Počiatočný atribút: $A1 \in \{0,1,2,3,4,5,6,7,8,9,10\}$ počet tried 4

Min 0, max 10, interval i=2,5 potom 0,1,2 priradí do triedy 1, 3,4,5 do triedy 2,

6 a 7 do triedy 3 a hodnoty 8,9,10 do triedy 4.

Výsledný atribút: $A1p \in \{$ tr1,tr1,tr1,tr2,tr2,tr2,tr3,tr3,tr4,tr4,tr4$\}$

FrequencyDiscretization - táto funkcia rozdelí dáta na určitý počet tried. V tomto prípade je počet tried vstupný atribút funkcie, ale ich počet musí byť rovnaký. Dáta sú rozdelené podľa veľkosti. Určí koľko je prvkov v množine dát a tento počet vydelí počtom tried a dostane počet prvkov v danej triede. Priradzuje túto triedu od najmenšieho prvku. Zistí, koľko jej ostalo prvkov a na koľko tried ju musí rozdeliť. Vydelí počet ostávajúcich prvkov ostávajúcim počtom tried a dostaneme počet prvkov. Toto stále opakuje, až rozdelí všetky prvky do tried. [6]

Príklad:

Počiatočný atribút: $A1 \in \{0,1,1,9,4,5,6,6,7,9\ \}$ počet tried 3.

Počet prvkov 10 počet tried 3. Počet prvkov v prvej triede 10/3=3 zvyšok 7. Do

prvej triedy budú priradené prvky 0,1,1. Výpočet druhej triedy: 7/2=3. Do druhej triedy

budú priradené prvky 4,5,6. Výpočet tretej triedy 4/1=4. Do tretej triedy budú

priradené prvky 9,6,7,9.

Výsledný atribút: $A1p \in \{$tr1,tr1,tr1,tr3,tr2,tr2,tr2,tr3,tr3,tr3$\}$.

MinimumEntropyPartitioning – funkcia rozdelí atribút tak, aby jednotlivé triedy mali čo najmenšiu entropiu. Funkcia má pracovať s numerickými dátami, ale pokiaľ majú atribúty numerické hodnoty, nedá sa spustiť. Ak použijeme atribúty s nominálnymi hodnotami, funkcia začne pracovať a po krátkom čase vytvorí chybové hlásenie- proces neúspešný. [6]

UserBasedDiscretization – funkcia rozdelí dáta na určitý počet tried. Vstupný atribút funkcie sú mená tried a ku každej triede je priradený horný limit priradenia. Prvky atribútu rozdelí do tried

podľa horného limitu. Ak sú niektoré prvky vyššie než je najvyšší limit, priradí tieto prvky do triedy tr"(počet tried)".[6]

Príklad:

Počiatočný atribút: $A1 \in \{0,1,1,9,4,5,6,6,7,9\}$

Triedy: „prvá" horný limit 2; „druhá" h.l. 6; „tretia" h.l 8.

Výsledný atribút:

$A1p \in \{$prvý,prvý,prvý,tr4,druhá,druhá,druhá,druhá,tretia,tr4$\}$

ExampleFilter - filtrácia dát podľa druhu atribútu, ktorý je v parametroch funkcie. Filtrujeme napríklad prvky, ktoré nemajú hodnotu. Z dát vynechá prvky, ktoré hodnotu neobsahujú. [6]

3 Spracovávané súbory

Systém tejto bakalárskej práce spracúva dva typy súborov, a to *arff* a veľmi známy formát *csv*. Štruktúra hlavičiek týchto súborov je veľmi rozdielna, preto si ich v nasledujúcej časti popíšeme.

3.1 Súbor *csv*

Csv – comma-separated values (hodnoty oddelené čiarkami), je jednoduchý súborový formát určený pre výmenu tabuľkových dát. Nie všetky tieto súbory majú ale rovnakú štruktúru, lebo pre *csv* neexistuje žiadny presne špecifikovaný formát. V praxi sa pojem *csv* vzťahuje na súbory, ktoré:

- obsahujú istý text (plain text) s kódovaním napríklad *ASCII*, *Unicode*

- pozostávajú zo záznamov (väčšinou jeden záznam na riadok)

- majú záznamy rozdelené do polí (stĺpcov), oddelených jedným rezervovaným znakom ako čiarka, bodkočiarka alebo tabulátor,

- v každom zázname obsahujú rovnaký počet stĺpcov.

V našom prípade potrebujeme, aby súbor v prvom riadku obsahoval názvy atribútov, v druhom riadku sa budú nachádzať dátové typy a potom budú nasledovať jednotlivé dáta.

```
outlook,temperature,humidity
string,real,real
sunny,25,90
rainy,15,75
..
```

3.2 Súbor arff

ARFF attribute-relation file format - je to súbor, ktorý popisuje zoznam atribútov. Tento formát bol vyvinutý na univerzite *Waikato,* primárne pre nástroj Weka.

Tento súbor má dve rozdielne sekcie. Prvá sekcia je hlavička, ktorá obsahuje názov relácie, zoznam atribútov a ich typov. Pred reláciu a atribút sa vždy musí dať znak @.

```
@relation weather
% These data were measured at the station in Ostrava
@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
```

Pri nominálnom type sú uvedené v zložených zátvorkách aj hodnoty, ktoré môže daný atribút nadobúdať. Preto je tento formát ľahší na spracovanie. Ďalej sa môže v hlavičke nachádzať komentár, pred ktorý dávame *%*.

Druhá sekcia obsahuje už samotné dáta. Táto sekcia vždy začína rezervovaným slovom *@DATA*. Jednotlivé dáta sú oddelené čiarkami a majú ten istý formát, ako dáta vo formáte *csv*.

```
@data

sunny,85,85
sunny,80,90
overcast,83
..
```

4 Implementácia systému

Zadanie bakalárskej práce určuje, že aplikácia má byť realizovaná pomocou technológie spoločnosti Microsoft .NET a programovacieho jazyka C#. Pri implementácii bol použitý .NET framework 4.5 a najnovšie vývojové prostredie Microsoft Visual Studio 2012.

4.1 Štruktúra systému

Celé jadro systému je postavené na rozhraní Windows Forms. Tu sa odohráva takmer celý beh programu. Od čítania súboru, cez jeho spracovanie, načítanie pluginov až po výstup. Vstupom pre systém sú textové súbory a výstupom predspracované dáta, ktoré užívateľ môže ale nemusí zobraziť.

4.2 Trieda *Attribut*

Táto trieda je pre celý systém kľúčová. Obsahuje všetky informácie o načítanom súbore a do jej objektu sa ukladajú aj informácie po aplikovaní nejakého pluginu. Všetky dátové členy sú reprezentované pomocou vlastností, teda pomocou ***get, set***.

- **filePath –** aktuálne umiestnenie súboru
- **index –** index daného atribútu v celom súbore
- **relationName –** názov relácie, v ktorej sa atribút nachádza
- **Name –** názov atribútu, ktorý je vždy unikátny
- **Count –** počet prvkov v danej relácii
- **Type –** dátový typ atribútu
- **Sum –** súčet číselných prvkov atribútu
- **Max –** maximálna hodnota číselného atribútu
- **Min –** minimálna hodnota číselného atribútu
- **Average –** priemer pri číselnom atribúte

Ďalej sa v tejto triede vytvára objekt typu ***Values***, ktorý je určený pre atribúty s nominálnou hodnotou. Do inštancie tejto triedy sa ukladajú jednotlivé názvy hodnôt atribútov **Name** a ich výskyt **Count**.

Trieda ***Attribut*** obsahuje jedinú metódu a to **Clone()**, ktorá robí hlbokú kópiu objektu, pretože je potrebné vytvoriť nový objekt s rovnakými dátami.

4.3 Načítavanie metadát

Čítanie dát zo súboru a ich následné spracovanie bolo v našom prípade najzložitejšou a časovo najnáročnejšou operáciou. Keďže formáty, ktoré spracovávame, majú rozdielnu štruktúru, na každý sme museli použiť iný postup.

Pri oboch súboroch bola použitá trieda ***BackgroundWorker***. Pomocou tejto triedy môže užívateľ sledovať priebeh čítania a aplikácia sa nebude javiť ako zamrznutá.

4.3.1 Súbor *arff*

Tento súbor má výhodu, pretože obsahuje hlavičku. Tým pádom si po prečítaní hlavičky dokážeme vytvoriť všetky potrebné informácie o dátach v súbore. Najmä to, aké hodnoty obsahujú nominálne atribúty. Všetky informácie budeme ukladať do objektu typu ***Attribut***.

Začneme čítaním hlavičky. Z prvého riadku získame názov relácie. Urobíme to tak, že skontrolujeme či riadok obsahuje „*@Relation*". Ďalej kontrolujeme či riadok obsahuje „*@attribute*", ak áno, tak skontrolujeme či obsahuje *„{"* – to je znakom toho, že atribút je nominálny a budú sa tam nachádzať jeho hodnoty, tie získame a do inštancie triedy ***Attribut*** predáme názov relácie, typ atribútu, meno atribútu a do listu typu ***Values*** pridáme hodnoty, ktoré sa nachádzajú medzi zloženými zátvorkami. V tom prípade, že riadok neobsahuje *„{"* znamená to že ide o numerický typ, tak do inštancie pridáme názov relácie, typ atribútu, meno atribútu a do listu typu ***Values*** pridáme *min, max, avg*.

Po získaní informácií z hlavičky budeme čítať celý súbor riadok po riadku až do konca. Jednotlivé riadky rozdelíme do poľa pomocou metódy **Split(',').** Potom pomocou cyklu prechádzame vytvorený objekt a kontrolujeme ho. Ak má byť nominálneho typu, tak podľa jeho názvu inkrementujeme danú hodnotu v objekte. Pokiaľ je hodnota typu numerického, pripočítame jej veľkosť a inkrementujeme počet kvôli výpočtu priemeru. Ďalej tu zisťujeme minimum a maximum daného atribútu.

4.3.2 Súbor *csv*

Nevýhodou tohto súboru je, že hlavička neobsahuje všetky informácie o jednotlivých atribútoch. Názov relácie získame z názvu súboru. V prvom riadku sú názvy jednotlivých atribútov a v druhom ich typ. Pri nominálnych dátach musíme počas čítania súboru kontrolovať či sa tam už daná hodnota atribútu nenachádza. Samotné čítanie už prebieha tak isto ako pri súbore *arff*.

4.3.3 Čítanie veľkých súborov

Ak užívateľ zvolí súbor, ktorý je väčší ako 100 MB, systém ho nemusí čítať celý. Môže si vybrať, ktoré atribúty chce prečítať a čítanie môže kedykoľvek zastaviť. Systém najprv spracuje hlavičku súboru, aby si užívateľ mohol vybrať, ktoré dáta chce čítať. Následne prechádza súbor a číta len tie dáta, ktoré označil užívateľ. Pri čítaní zároveň zapisuje jednotlivé dáta do nového súboru- následné spracovanie bude oveľa jednoduchšie. Každou iteráciou inkrementujeme pomocnú premennú a zobrazíme ju užívateľovi, aby videl, koľko riadkov už systém spracoval.

4.4 Implementácia rozhrania

Bakalárska práca určuje, že systém má byť modulárny, takže bude realizovaný pomocou zásuvných modulov – pluginov. To znamená, že každý jeden zásuvný modul musí implementovať rozhranie, ktoré určuje, aké dátové členy a členské metódy by mal obsahovať.

4.4.1 Návrh rozhrania

Každý plugin používa na komunikáciu s hlavným programom veľmi jednoduché rozhranie **IPlugin**, ktoré implementuje len 3 členy.

```
public interface IPlugin
{
      string name { get; }

      System.Windows.Forms.UserControl MainInterface { get; }
      List<Attribut> getData(List<Attribut>
      selectedColumns,List<Attribut>dataList);
}
```

Zdrojový kód hlavného rozhrania

- **name –** táto vlastnosť vracia názov pluginu. Pomocou nej zobrazujeme meno pluginu a následne hlavný program vie, ktorý plugin má zavolať.

- **MainInterface –** keďže jednotlivé pluginy sú reprezentované pomocou komponenty UserControl(UC), tento člen vracia práve daný UC a hlavný program ho zobrazí.

- **getData –** táto metóda je kľúčová pre celý plugin. Pomocou nej získame z hlavného programu všetky metadáta a metadáta, ktoré majú byť spracované. Plugin ich následne spracuje a vráti hlavnému programu.

4.5 Zobrazovanie dát a spätná rekonštrukcia

Načítané dáta zobrazujeme pomocou komponenty DataGridView. Postup je veľmi jednoduchý. Z objektu typu ***Attribut*** najprv vytvoríme hlavičku dát a potom pridávame jednotlivé dáta čítaním súboru.

Keďže každý jeden plugin vytvára dočasný súbor, spätná rekonštrukcia je veľmi jednoduchá. Do listu si ukladáme cesty k daným súborom a potom, už ich len jednoducho načítame a zobrazíme.

4.6 Ukladanie scenára spracovania

Scenár spracovania je postup, pri ktorom sa zaznamenáva, na aký atribút bol použitý ten ktorý plugin. Keďže tento scenár potrebujeme uchovávať aj po ukončení aplikácie, použili sme binárnu serializáciu, ktorá uloží do binárneho súboru aktuálny stav objektu. V našom prípade to bude inštancia špeciálne navrhnutej triedy *MultiMap<V>*, kde ukladáme potrebné informácie.

4.6.1 Trieda *MutliMap<V>*

Princíp tejto triedy je veľmi jednoduchý. Keďže pluginu môžeme predať aj viac ako jeden atribút, potrebujeme nejakú štruktúru, ktorá bude pre jeden kľúč reprezentovať viac hodnôt. V tomto prípade plugin bude kľúč a zvolené atribúty budú hodnoty.

Táto trieda používa *Dictionary <string,List<V>>*. To znamená, že používa kľúč typu string a hodnota bude list zvoleného typu. Z tejto triedy si ukážeme implementáciu metódy, ktorá pridá kľúč aj s jeho hodnotami.

```
public void Add(string key, V value)
{
      List<V> list;
      if (this._dictionary.TryGetValue(key, out list))
      {
          list.Add(value);
      }
      else
      {
          list = new List<V>();
          list.Add(value);
          this._dictionary[key] = list;}}
```

4.7 Práca s pluginmi

Platforma .NET je vďaka svojej architektúre a jednoduchosti pri práci s knižnicami .dll priam ideálna na vytváranie aplikácií modulárne. Keďže systém má byť modulárny, potrebujeme pracovať s pluginmi, a to nám umožní trieda *PluginService.* V nasledujúcej časti si rozoberieme niektoré jej časti, aby sme lepšie pochopili ako funguje.

4.7.1 Vyhľadávanie pluginov

Metóda **FindPlugins(string path)** je veľmi jednoduchá metóda, ktorej argument prijíma cestu k adresáru, kde by mali byť uložené dostupné pluginy. Prechádza všetky súbory a kontroluje príponu „**.dll* “. Následne ich uloží do kolekcie.

```
public void FindPlugins(string Path)
{
       colAvailablePlugins.Clear();

       foreach (string fileOn in Directory.GetFiles(Path))
       {
             FileInfo file = new FileInfo(fileOn);
```

```
            if (file.Extension.Equals(".dll"))
            {
                this.AddPlugin(fileOn);
            }
        }
    }
```

4.7.2 Načítavanie pluginov

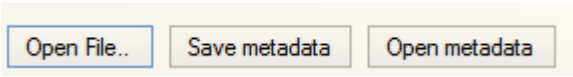
V metóde **addPlugin(string fileName)**, ktorá prijíma ako argument plugin, ktorý by mal byť načítaný, vytvoríme nové zostavenie pluginu. Ďalej budeme prechádzať všetky typy, ktoré obsahuje konkrétny plugin. Pretože nechceme, aby sme načítali nejaký plugin, ktorý naša aplikácia nebude podporovať. Skontrolujeme či, daný plugin implementuje naše hlavné rozhranie a v podmienke *if* je kontrola atribútov daného typu načítaného zostavenia.

4.8 Užívateľské rozhranie

Užívateľské rozhranie je implementované pomocou .NET API Windows Forms. Bolo použité pre jeho jednoduchosť a prehľadnosť. Ako predloha bola použitá aplikácia Weka.

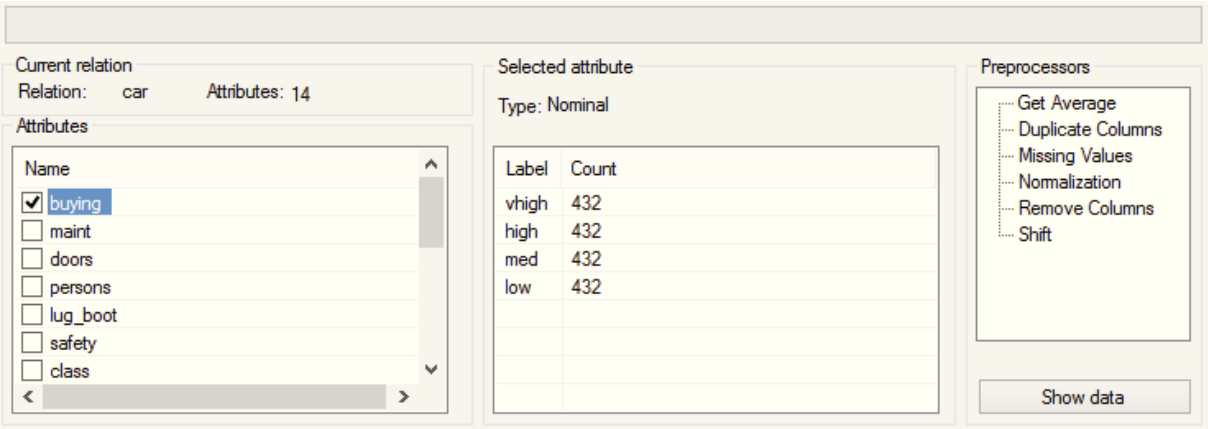
4.8.1 Menu

Menu obsahuje len tri položky. **Open file** otvoríme požadovaný súbor. **Save metada** uloží získané metadáta a **Open metada** otvorí uložené metadáta.



Obrázok 2: Menu

4.8.2 **Zobrazenie metadát**



Obrázok 3: Zobrazenie metadát a možnosť ich spracovania

V tejto časti aplikácie sa zobrazia informácie o danom súbore. V **Current relation** je zobrazený názov aktuálnej relácie (**Relation)** a počet jej atribútov (**Attributes**). Proces jeho načítania užívateľovi zobrazuje komponenta ProggressBar.

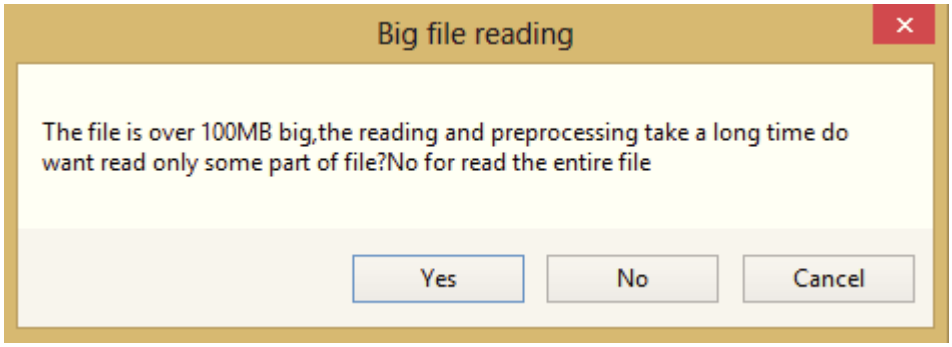
Po kliknutí na nejaký atribút sa v **Selected attribute** zobrazí typ daného atribútu(**Type)** a v komponente ListView jednotlivé informácie. Pri numerickom type je to *maximum, minimum* a *priemer*. Pri nominálnych dátach počet výskytov jednotlivých hodnôt.

Po zaškrtnutí komponenty CheckBox sa v časti **Preprocessors** odblokujú jednotlivé pluginy, ktoré po dvojkliku spracujú zvolené atribúty. Ak sa užívateľ bude snažiť urobiť neprijateľnú operáciu, ako napríklad spriemerovať nominálne atribúty, bude nato upozornený dialógovým oknom. Aplikácia ponúka tieto metódy spracovania:

- **Get Average –** vráti celkový priemer zvolených atribútov
- **Duplicate Columns –** jednoducho pridá zvolené atribúty ešte raz
- **Missing Values** – doplní chýbajúce hodnoty buď priemerom alebo hodnotou s najväčším výskytom.
- **Normalization -** číselné hodnoty prepočíta do intervalu <0,1>
- **Remove Columns –** odstráni zvolené atribúty z celého dokumentu
- **Shift –** posun hodnôt o určitú hodnotu

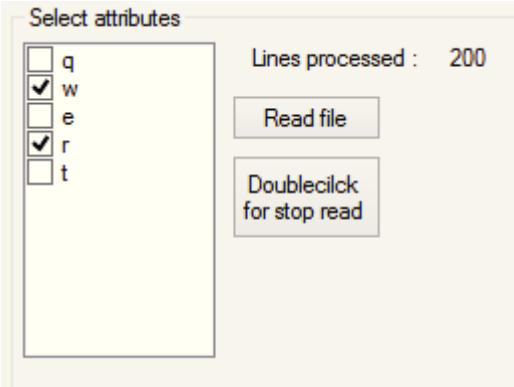
4.8.3 Čítanie veľkých súborov

Ak je veľkosť súboru väčšia ako 100 MB, dá aplikácia užívateľovi na výber či chce čítať celý súbor alebo len časť súboru.



Obrázok 4: Voľba pri veľkom súbore

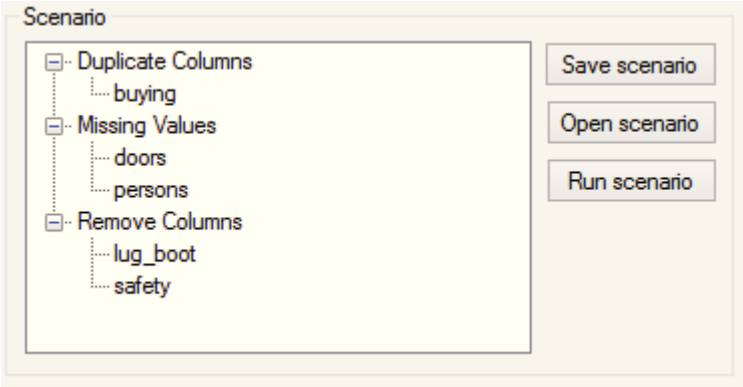
Pri kliknutí na **No** pokračuje program klasickým spôsobom načítania. Po kliknutí na **Yes** dá program užívateľovi túto možnosť:



Obrázok 5: Výber atribútov

Užívateľ si môže vybrať, ktoré atribúty chce načítať. Po zvolení potrebných atribútov užívateľ klikne na **Read file.** Aplikácia ukazuje, koľko riadkov už má spracovaných a po dvojkliku na **Stop read** sa čítanie ukončí a program sa prepne do hlavného okna na spracovanie.

4.8.4 Scenár spracovania

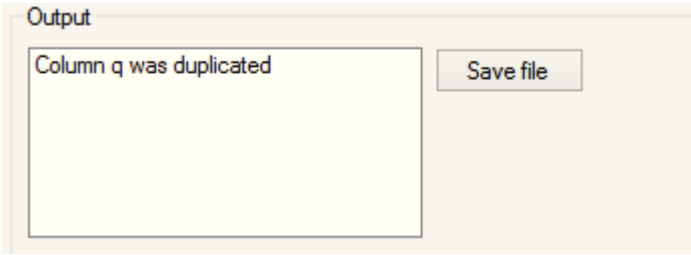


Obrázok 6: Scenár spracovania

Po každom aplikovanom plugine sa automaticky vytvorí scenár, ktorý zaznamenáva, ktorý atribút bol označený a aká operácia bola na ňom vykonaná. Užívateľ si potom môže daný scenár uložiť (**Save scenario**) a neskôr otvoriť (**Open Scenario**) a spustiť (**Run scenario**). Systém potom automaticky vykoná operácie na danom súbore.

4.8.5 Plugin

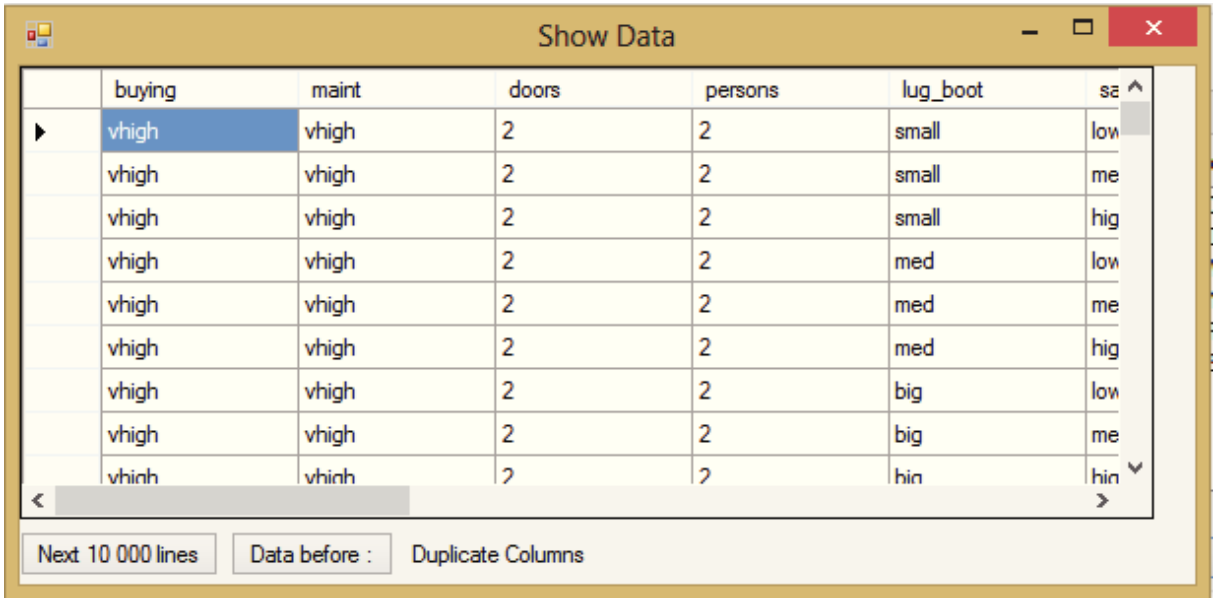
Uživateľské rozhranie každého pluginu je veľmi jednoduché, obsahuje len výstup o operácii a možnosť uloženia súboru (**Save file**).



Obrázok 7: Výstup pluginu

4.8.6 Zobrazenie dát

Dáta zobrazujeme pomocou komponenty DataGridView. Vždy sa zobrazí prvých 10 000 riadkov. Užívateľ môže pomocou kliknutia na **Next 10 000 lines** pohybovať v súbore. Pomocu **Data before** môže zobraziť históriu dát počas spracovávania.



Obrázok 8: Zobrazenie dát

4.9 Implementácia pluginov

V tejto časti práce si ukážeme implementáciu jednotlivých pluginov. Budú tu uvedené len najdôležitejšie algoritmy, ktoré reprezentujú jednotlivé metódy predspracovania.

Pluginy sú vytvorené pomocou projektu Class Library, teda každý bude mať príponu *.dll. Všetky pluginy sú vytvorené ako User Control, kde sa nachádza ovládanie a výstup. Každý jeden plugin si po svojom zavolaní uloží metadáta a metadáta, ktoré má spracovať. Následne si zistí, o aký typ súboru ide, pretože súbor *arff* a *csv* majú rozdielne hlavičky a preto ich treba ináč spracovať. Pri súbore *arff* sa vždy vytvorí hlavička podľa zvoleného pluginu.

```
public List<Attribut> getData(List<Attribut> a, List<Attribut> b)
{
     for (int i = 0; i < a.Count; i++)
           if (a[i].Type1 == "STRING")
    {
                 MessageBox.Show("Column " + a[i].Name1 + " is not
numeric ");
                 return b;
    }
     foreach (var item in a)
            columnsList.Add(item);
```

```
        foreach (var item in b)
        {
                dataList.Add(item);
        }
        getFile();
        return dataList1;
}
```

4.9.1 Plugin na duplikáciu

Tento plugin zduplikuje zvolené stĺpce (atribúty). Pridá ich na koniec súboru v každom riadku. Pritom musí vygenerovať nové unikátne meno novovytvoreného atribútu. Spracovanie prebieha v obidvoch formátoch rovnako. Pomocou metódy **Split(‘,’)** si rozdelíme celý riadok do poľa a potom porovnávame jeho indexy s indexmi zvolených atribútov. Ak sa indexy zhodujú tak danú hodnotu pridáme na koniec riadku.

```
protected string getDuplicateValue(string line)
{
        string[] data = line.Split(',');
        for (int i = 0; i < columnsList.Count; i++)
        {
            line += "," + data[columnsList[i].Index];
        }
        return line;
}
```

Algoritmus na duplikovanie

4.9.2 Plugin na odstránenie atribútov

Tento plugin odstráni zo súboru zvolené atribúty. Postup je veľmi jednoduchý. Riadok si rozdelíme do poľa a potom prechádzame zvolené stĺpce. Pokiaľ sa daný index rovná indexu v poli, zväčšíme pomocnú premennú. Ak sa táto pomocná premenná pri kontrole nerovná nule znamená to, že tento prvok je určený na odstránenie – teda ho nezapíšeme.

```
protected string removeColumn(string line)
{
     string[] data = line.Split(',');
     short tmp;
     line = "";
     for (int i = 0; i < data.Length; i++)
          {
               tmp = 0;
               foreach(var item in columnsList)
               {
                    if (i == item.Index)
                       tmp++;
               }
               if (tmp == 0)
                    line += data[i] + ",";
          }
     return line;
}
```

4.9.3 Plugin na doplnenie chýbajúcich hodnôt

Tento plugin má za úlohu doplniť chýbajúce hodnoty. Pri číselnom type sa za chýbajúcu hodnotu dosadí hodnota priemerná a pri nominálnom type hodnota atribútu, ktorý má najväčší počet výskytov. Pomocou metódy **Split(‘,’)** si riadok rozdelíme do poľa. Potom pomocou indexu zadaného atribútu skontrolujeme či sa danej pozícii v poli nachádza **„?“** , **„“** , **„ “** ak áno tak ho zameníme za potrebnú hodnotu.

```
protected string addValue(string line)
{
      if (line.IndexOf(",", line.Length - 2) == line.Length-1)
           line = line.TrimEnd(line[line.Length - 1]);

      string[] data = line.Split(',');
      foreach (var item in columnsList)
      {
    if ((data[item.Index] == "?") || (data[item.Index] == "") ||
(data[item.Index] == " "))
          {
     if (item.Type1 == "REAL" || item.Type1 == "DOUBLE")
                {
                      data[item.Index] =
dataList[item.Index].Average.ToString();
                }
                else
                {
                      data[item.Index] = getMax(item.Index);
                      newData(item.Index);
                }
   }
      }
       line = "";
       for (int i = 0; i < data.Length; i++)
           line += data[i] + ",";

      return line;
}
```

Metóda na odstránenie chýbajúcich hodnôt

4.9.4 Plugin na normalizáciu

Tento plugin slúži na normalizáciu číselných hodnôt. To znamená, že hodnoty prepočíta do intervalu <0,1>. Postupovať budeme tak, že si riadok pomocou **Split(‘,’)** rozdelíme do poľa a podľa indexu zvolených atribútov postupne normalizujeme podľa požadovaného vzorca. V prípade, žeby hodnota chýbala, kontrolujeme či sa dá hodnota v poli konvertovať na číslo.

```
protected string normalizeValue(string line)
{
        if (line.IndexOf(",", line.Length - 2) == line.Length - 1)
           line = line.TrimEnd(line[line.Length - 1]);
        double tmp = 0;
        string[] data = line.Split(',');
        foreach (var item in columnsList)
        {
           bool canConvert = double.TryParse(data[item.Index],out
tmp);
           if(canConvert)
data[item.Index] = ((Double.Parse(data[item.Index]) - item.Min) /
(item.Max - item.Min)).ToString();
        }
        line = "";
        for (int i=0; i < data.Length; i++)
            line += data[i]+",";

        return line;
}
```

Metóda na normalizáciu

4.9.5 Plugin na počítanie priemeru

Tento plugin vypočíta priemer zo zadaných stĺpcov a to tak, že prechádzame celý súbor, sčítame všetky hodnoty zvolených stĺpcov a vydelíme ich počtom. Keďže počet jednotlivých prvkov už máme v metadátach, netreba ho opäť zisťovať.

```
protected double getAverage(double sum)
{
       double count = 0;
       foreach (var item in pluginList)
       {
             count += item.Count;
       }
       return sum/count;
}
```

Metóda na počítanie priemeru

4.9.6 **Plugin na posun**

Tento plugin posunie hodnoty zvoleného stĺpca o danú hodnotu. Prechádzame celý súbor a ku každej hodnote pripočítame užívateľom zadanú hodnotu.

```
protected string shiftValue(string line)
{
    if (line.IndexOf(",", line.Length - 2) == line.Length - 1)
         line = line.TrimEnd(line[line.Length - 1]);
    double tmp = 0;
    string[] data = line.Split(',')
    foreach (var item in columnsList)
    {
       bool canConvert = double.TryParse(data[item.Index], out tmp);
       if (canConvert)
         data[item.Index] =(double.Parse(data[item.Index]) +
shift).ToString();
            }
   line = "";
   for (int i = 0; i < data.Length; i++)
       line += data[i] + ",";
```

```
    return line
}
```

Metóda na posun

Záver

V tejto bakalárskej práci som zdokumentoval niektoré metódy na predspracovanie dát, ktoré sa dnes najčastejšie používajú. Následne som vytvoril desktopovú aplikáciu, ktorá niektoré tieto metódy používa a je schopná ich aplikovať na dva typy textových súborov.

Ďalším vývojom tohto nástroja by mohlo byť zakomponovanie ďalších metód na predspracovanie a následne by mohol tento systém umožniť aj samotné dolovanie z dát.

Jednou z výhod tohto nástroja je, že dokáže spracovať aj veľké súbory, na rozdiel od iných , ktoré jednotlivé dáta ukladajú do operačnej pamäti a tým pádom nemajú toľko priestoru.

Počas spracovávania tejto práce som zistil, že predspracovanie dát je veľmi dôležitý proces, pri práci s dátami a je oveľa náročnejšie ako následné modelovanie výsledkov, pretože v dnešnej dobe, kedy sme priam zahltený dátami a nie všetky dáta sú potrebné alebo korektné. A z ich nárastom budeme potrebovať ich elimináciu, preto má predspracovanie budúcnosť.

Použitá literatúra

[1] BERKA, Petr. *Dobývání znalostí z databází*. Vyd. 1. Praha: Academia, 2003, 366 s.

ISBN 80-200-1062-9.

[2] Famili, A. et al. **Data Preprocessing and Intelligent Data Analysis**. *Intelligent*

data Analysis 1 (1997) , page 3-23, Elsevier Science B.V.

[3] **Introduction to missing data** online [cit. 01.05.2013] Dostupné z

http://missingdata.lshtm.ac.uk/index.php?option=com_content&view=section&id=7&Itemid=96

[4] Theodoridis S., Koutroumbas K. **Pattern recognition second edition**.
London : Academic Press, 2003. ISBN 0-12-685875-6

[5] **Smerodajná odchýlka,** Wikipedia online [cit. 01.05.2013].Dostupné z

http://en.wikipedia.org/wiki/Standard_deviation

[6] Rapid-I GmbH. *User Guide* 2007, posledná revízia 31.6.2007
Dostupné z: http://www.rapidminer.com/

Prílohy

Obsah priloženého CD obsahuje

- Text bakalárskej práce vo formáte pdf.
- Priečinok s názvom BP, v ktorom je MS Visual Studio solution s aplikáciou.